Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds.

REAL-TIME SPATIO-TEMPORAL DATABASES: BRIDGING THE GAP BETWEEN EXPERIMENTABLE DIGITAL TWINS AND DATABASES

Moritz Alfrink
Jürgen Roßmann

Institute for Man-Machine Interaction
RWTH Aachen University
Templergraben 55,
Aachen, NRW 52074, GERMANY

ABSTRACT

Digital Twins (DTs) have come to be an indispensable methodology in the fields of robotics, autonomous driving, sensor simulation, Hardware-in-the-loop (HIL), Augmented reality (AR) and a variety of other fields. In this paper, we argue why databases supporting the three-dimensional (3-D) simulation of Experimentable Digital Twins (EDTs) have strict criteria, including real-time capabilities, spatial organizability of data, and temporal awareness in data representation. We thus propose the Real-Time Spatio-Temporal Database (RTSTDB) as an architecture combining these capabilities and thus meeting all EDT requirements. To facilitate real-time operation, our implementation of this concept utilizes the highly parallelized vector capabilities of Graphics processing units (GPUs). We show the successful application of our approach by providing two use cases that exhibit the identified requirements. Finally, we define a standardization for RTSTDB interfaces to maximize interoperability.

1 INTRODUCTION

DTs are core component of today's industrial product life cycle management. The extension of the DT with 3-D simulations - the EDT - introduced by Jürgen and Schluse (2020), makes it possible to construct virtual prototypes and assess their performance in virtual testbeds before even a single real asset is built, as illustrated in Figure 1. This does not only reduce the costs tremendously but also minimizes the development time. To drive the virtual testbed a powerful simulation is essential. The main component, storing all information of a simulation, is its database. Being the core module, it must deliver all functionalities required for all simulation tasks EDTs need. Our primary contribution is the collection of the most crucial capabilities needed to simulate EDTs, integrating them into a single RTSTDB architecture, and applying them to two use cases that put those capabilities to the test. For this purpose, the following sections aims to identify requirements posed by EDTs.

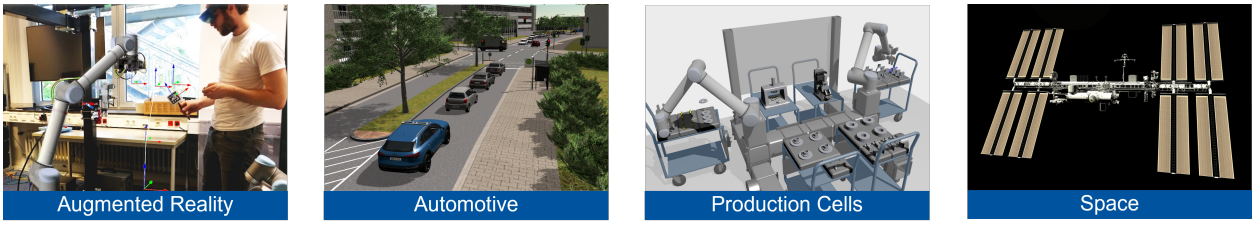

Figure 1: Different simulations of EDTs.

2 THEORY

This part aims to derive requirements DTs set to databases, the requirements are collected from related work from the fields of robotics, autonomous driving, sensor simulation and Extended reality (XR) applications.

The DT model has its origin in a conceptual idea of Grieves (2002) for Product-Lifecycle-Management (PLM). The goal was to model the entire life-cycle of a product from its initial design, over manufacturing and finally its disposal. Grieves dedicates the Real Space as data supply for the Virtual Space, which in exchange provides augmented information and process to the Real Space. While this was originally referenced as Information Mirroring Model, the term DT was attached to this model in Grieves (2011). A definition of (Grieves 2016) states "the DT is a set of virtual information constructs that fully describes a potential or actual physically manufactured product from the micro atomic level to the macro geometrical level. At its optimum, any information that could be obtained from inspecting a physical manufactured product can be obtained from its Digital Twin". This is further specified to a fully annotated 3-D model, bill of materials, bill of process, bill of services, and bill of disposal forming the Digital Twin Prototype (DTP) which can be seen as a template for DT instances. But the concept of DTs is not limited to plain information storing or mirroring, EDTs as instances in an active 3-D simulation can provide additional value for knowledge discovery as shown by Schluse et al. (2018). Their work also underlines the importance of real-time ability of the simulation framework. This comes into play in HIL applications, where the simulation must work in a loop with real hardware to validate theoretical models from the engineering phase as shown by Benoit et al. (2017) and Rahnamai and Rowlands (2017). The real-time constraints are justified by the fact that the simulation must provide the data rates, which the hardware requires to function as well as the output data, it feeds back into the simulation. In Virtual reality (VR) and AR applications, with humans in the loop, Lee (2021) states that interactive applications require at least 90 frames per second (fps) to enhance the immersion and prevent motion-sickness.

Another perspective on EDTs shows Thieling et al. (2021), by incorporating the aspect dynamic properties of EDTs such as velocities in their radio detection and ranging (RADAR) simulation. Emde and Rossmann (2013) further performs the simulation aware of the timings in sensor scanning pattern as well as dynamic motion of the target and the sensor itself. This can be illustrated with a laser scanner mounted to a car. To scan the entire environment, the laser ray is directed circular on multiple height level by a rotating and tilting mirror. During the time it takes to complete one scan, the sensor' ray has completed multiple revolutions and the car as well as the other road users have moved. The temporal behavior is to be incorporated into the sensor simulation and thereby into the database.

The aforementioned properties of EDTs lead to the following requirements for the database:

1. Real-Time: The database must be able to process transactions within hard dead-lines.
2. Spatiality: The database must organize the data in a way, that complex spatial-relation queries can be answered efficiently. Furthermore, the database must be able to represent the DT with its geometry, pose and material properties.
3. Temporality: The database must be able to represent the temporal changes in the data. For the DT this includes primary its dynamic properties.

3 RELATED WORK

The list of requirements, set up in Section 2, is to be tested against existing database concepts of real-time, spatiality, and temporality. Beginning with real-time database requirements, Lin and Son (1994) characterize Real-Time Database (RTDB) by explicitly posing timing constraints on transaction, such as deadlines. A transaction is a group of create, read, update, and delete (CRUD) operations, that have the following properties: atomic, consistent, isolated, and durable (ACID) defined by Balusamy et al. (2021). Atomic is a property that declares each transaction as one unit of operations where either all or none are executed. Consistency is a property that ensures, that the database remains in a consistent state after a successful transaction. Isolation is a property that allows users to operate concurrently on data, without any conflicts.

Durability is a property that ensures that completed transactions are permanent even if the system crashes. An initial, loose definition given by Ester et al. (1999) describes a Spatial Database System (SDBS) as relational databases, explicitly storing spatial location and extension. This implicitly captures the spatial neighborhood as well. Shekhar et al. (1999) sharpen this definition by requiring extended functionality to process the spatial data types (called geometry or features). Tian, Ji, and Scholer (2015) provide a recent approach to answer queries for objects at a particular position during a period of time in Transportation Management Systems and geographic information system (GIS). Their system relies on constructing a Morton R-tree inside the relational database for fast data retrieval. Garg et al. (2016) explore how frequent updates in spatial databases can be handled efficiently. They propose to classify the data as moving or static to maintain performance. An early attempt to handle temporality is discussed by Kuroki, Makinouchi, and Ishizuka (1997) as data which is bound to time, that it either exists only at certain points in time or that its location is dependent on time. This has the also impact on their proposed query language, which has to incorporate timings for the query to be evaluated. Temporality is further discussed by Zheng et al. (2014) in the context of path planning for ships. To avoid collisions the planning algorithm has to incorporate the individual positions of the ships temporally aware. Yu et al. (2021) cover the field of data mining in air traffic flow statistics. Their approach is able to capture the features of complicated 3-D time-dependent airspace and their database is optimized to deal with big volume spatio-temporal data covering high-dimensional features.

The related work indicates that the real-time databases must complete ACID transaction within fixed deadlines, spatial databases must organize their data in a way that relations in space can be determined efficiently and temporal databases must represent the behavior of their data over time. While all the stated literature provides partial solutions to the individual problems, none of them units all requirements into one database providing an holistic solution. This is where the RTSTDB comes into play.

4 ARCHITECTURE

In this section, we are going to outline the architecture and the interface of a RTSTDB. To meet the requirements established in Section 2. The start point for the introduction of the architecture is an overview of the usage of the RTSTDB together with a 3-D simulation as shown in Figure 2. The simulation consists of independent modules, with different execution times. Each module can access the RTSTDB via the standardized interface to perform transactions. The transactions consist of a single or multiple operations, respectively queries, to be executed with ACID guarantees. To meet real-time constraints as discussed in Section 3, queries can optionally include deadlines or priorities.

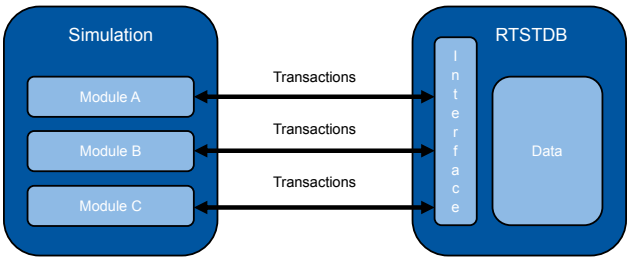

Figure 2: Architecture overview.

While creating and deleting are basic actions that ordinary databases can handle effortlessly, queries to select complex spatial and/or time-variable information, require a complex internal data structure to be answered efficiently. This in turn also means more complex update functions to ensure a consistent database state. The following list provides an overview of the basic query primitives that define the interface.

- Create, read, update, and delete: Basic query handle entities in the database.

- Ray-Tracing Query: Trace arbitrary rays in the environment and obtain the nearest entity hit. The Rays can be emitted from EDTs, inheriting their dynamic properties.
- Cone-Tracing Query: Trace arbitrary cones in the environment and obtain the nearest entity hit. The Cones can also be emitted from EDTs.
- Nearest Neighbor Query: Find the nearest entity to a given spatial location.
- Region Query: Collect all entities within a given area.
- Intersection Query: Collect all entities which intersect each other with their geometries hulls. Determine exact collision information, which contains position, normal, and depth of the intersections.
- Render Query: Generate an image for a certain point of view.

The data requires acceleration structures (ASs) in order to process the queries efficiently. However, before deciding on a suitable structure, we must make a key assumption that matches common usage patterns. The assumption is based on the fact that EDTs are often dynamic, despite the fact that their underlying geometry is largely static or can be remodeled as numerous dynamic EDTs with static geometries. This observation allows us to define two levels of ASs: The Top-Level Acceleration Structure (TLAS) and the Bottom-Level Acceleration Structures (BLAS). This practice is also common in ray-tracing applications. The TLAS operates on the EDT' pose combined with a broad enclosing hull, which we choose to be an axis-aligned bounding box (AABB). While the BLAS works at geometry level on either triangle, voxel or signed distance field (SDF) basis. Splitting into two levels gives the great advantage of only being required to update the TLAS when EDTs change their pose or one of the BLAS in case of geometry changes. Furthermore, allows the two-level AS to incorporate temporal properties of EDTs on a per-ray or per-cone basis while tracing (this is further elaborated in 5).

Before we describe how the queries are handled we introduce the required TLAS and BLAS shown in Figure 3.

4.1 Top-Level Acceleration Structure

We choose the TLAS to be a bounding volume hierarchy (BVH), since Karras (2012) showed that they can be built efficiently and in parallel on GPUs, while still archiving a high quality as shown by Karras and Aila (2013). Another property of BVHs is that they can be compressed into a hierarchy, in which each internal node holds N instead of two child nodes. According to Benthin et al. (2018), these so-called BVHNs can lead to a halving of the memory requirement, with little loss of performance. But the primary advantage of compressed BVHs is, that the stack required for tracing and nearest neighbor searches inside the GPU program can also be compressed. This is significant because GPUs have a finite amount of memory and registers per thread, therefore excessive register utilization results in low GPUs occupancy. Apart from short build times and memory efficient storage, the BVH can narrow a wide range of the proposed queries down to a set of EDTs, which must be tested for an exact solution. At its leaf nodes, the BVH keeps a reference to the EDT for this reason. This also enables for repeated instantiation of a geometry, as demonstrated in the BLAS section of Figure 3. We propose using a second local TLAS cache in addition to the BVH, which is the sole global TLAS that covers the full virtual scene. This cache is according to Bán and Valasek (2020) a cascade of SDFs of order 0, operating like clipmaps, shown in the top-right part of Figure 3. It allows to determine the distance from any point in the cascade to the nearest surface, whereat the sign determines whether the point is inside or outside the surface. The gradient of the SDF at the sample location approximates further the direction to the surface. The cache can be placed around a camera or in a region with a high number of EDTs.

4.2 Bottom-Level Acceleration Structures

Multiple types of ASs are employed in the BLAS to meet all of the requirements. Similar to the TLAS a BVHN is used to present each unique geometry. In addition to the BVH, an SDF of order 1 is used to store the geometries as illustrated in Figure 4. When compared to the order 0 SDF, the first order

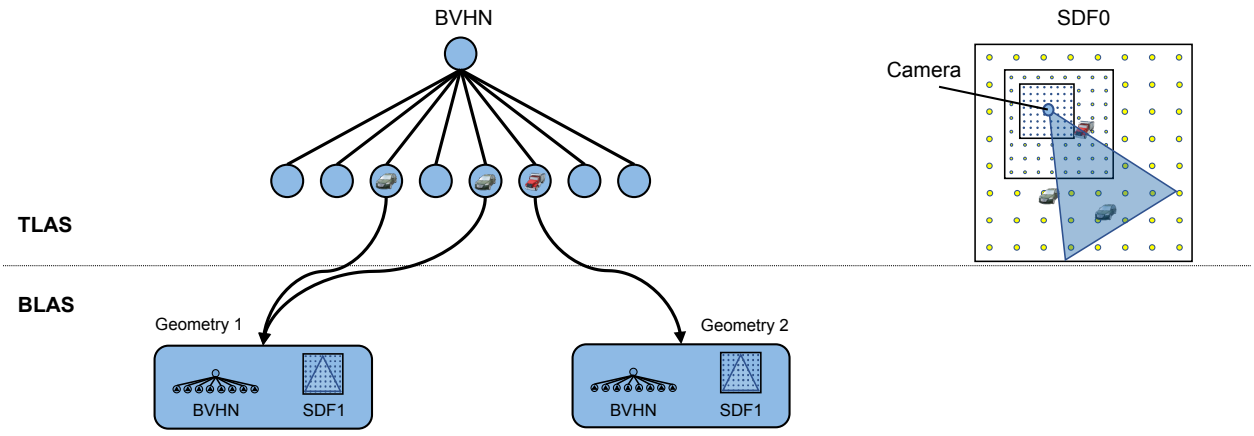

Figure 3: Top-and bottom-level acceleration structures.

SDF offers various superior qualities, including up to an order of magnitude reduced memory usage while maintaining an equivalent approximation error validated by Bán and Valasek (2020). Furthermore, the first order representation allows for the representation of surfaces with a grid resolution lower than that of the SDF grid. The reason for this is that the first order SDF, rather than only the signed distance, contains the plane equation of the nearest surface plane for each grid vertex.

4.3 Queries

After defining the ASs used as basis for the queries, we are going to describe how the queries make use of them.

The Ray-Tracing Query uses straightforward both BVH layers to determine intersections with the geometry of the EDTs in the scene. The Cone-Tracing Query leverages the top-level BVH to determine potential EDTs intersecting the cones' path. For each candidate, the first-order SDF of the geometry is used to verify or to falsify. The Nearest Neighbor Query is also based on the BVHs to determine the nearest of EDT or when required even the nearest triangle of an EDT. Region Query tests a region defined by a box or sphere shape against collision with the EDTs or when required the triangles of the EDTs. The Intersection Query uses the Region Query to determine potential pairs of intersecting entities in the broad phase. This is accomplished by testing the AABB of dynamic EDTs against the AABBs of static and dynamic EDTs in the scene for collision. The obtained pairs of potential colliders are in the narrow phase divided into convex-convex and concave-convex/concave pairs. The convex ones are tested for intersection using a method based on the GJK algorithm created by Gilbert, Johnson, and Keerthi (1987) in combination with Expanding Polytope Algorithm (EPA) developed by (van den Bergen 2003). The GJK algorithm requires only a support function to operate. This function returns the furthest vertex of the geometry in a given direction and can be accelerated using the geometries' BVH. For pairs with one or more concave geometries, the first order SDF is used to approximate the collision as shown by the general approach of Koschier et al. (2016). The Render Query is the most complex of the queries since it uses itself a combination of the other queries. In a nutshell for the first light bound a visibility query is performed determining all EDTs visible to the camera, which are then rasterized into the output image. To compute illuminance reaching each pixel in the image, shadow maps are leveraged to evaluate the occlusion term of the light sources. The second and above bounces of light caused by reflections, transmissions or absorption are handled using either Ray-Tracing Queries or Cone-Tracing depending on the roughness of the surface. The roughness is proportional to how much the beam fans out allowing them to be approximated by cones.

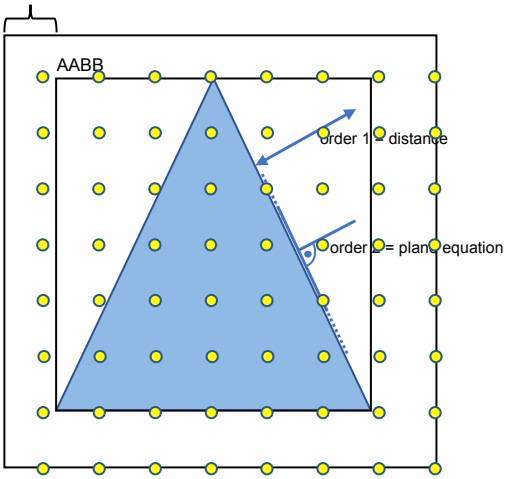

Figure 4: Signed distance field cascade.

5 IMPLEMENTATION

This section aims to give insight into the implementation details of the ASs and queries introduced in the previous Section 4. The current implementation aims to process as much as possible on GPUs to make use of their massive parallel computation power. The partition of this section is held similar to the previous, which sets the start point of the ASs.

To build the ASs on the GPU we are required to provide all the information in GPU memory, which leads to the definition of a general GPU data layout. The EDTs are characterized by their world-space pose given by position ${}^{\mathrm{WS}}\underline{x}$ and orientation ${}^{\mathrm{WS}}\underline{\theta}$, linear and angular velocity ${}^{\mathrm{WS}}\underline{\dot{x}}$ and ${}^{\mathrm{WS}}\underline{\dot{\theta}}$ respectively, scale $\underline{s}$, geometry, and material reference. To able to implicitly store the geometries' AABB in the pose matrix, the geometries' positions are stored in a normalized range of $[-1\ 1]$ and centered around the origin. To obtain the original mesh again, the positions must be scaled by $\underline{s}_{AABB}$ and transformed by $\underline{x}_{AABB}$. We choose to store the information as seen in Table 1, notice that the orientation information is redundant due to different representations required inside the GPU programs. Furthermore, the last row of pose matrix is being discarded because it is always similar to third row of the identity matrix.

Table 1: GPU representation of an EDT.

Variable	Representation	Space	Type	Quantisation
pose & scale	$\left[\mathrm{rot}({}^{\mathrm{WS}}\underline{\theta}) \cdot \mathrm{diag}(\underline{s}) \quad {}^{\mathrm{WS}}\underline{x}\right] \left[\mathrm{diag}(\underline{s}_{AABB}) \quad \underline{x}_{AABB}\right]$	$\mathbb{R}^{3x4}$	float	32bit
orientation	${}^{\mathrm{WS}}\underline{\theta}$	$\mathbb{R}^4$	float	32bit
linear velocity	${}^{\mathrm{WS}}\underline{\dot{x}}$	$\mathbb{R}^3$	float	32bit
angular velocity	${}^{\mathrm{WS}}\underline{\dot{\theta}}$	$\mathbb{R}^3$	float	32bit

Given the information in the GPU's memory the TLAS' and BLAS' BVHs can be built on the basis of Karras (2012) work. This means we start by assigning 63 bit Morton Codes to the centers of the EDTs' AABB. This gives a grid resolution for the AABB centers of $2^{21} \approx 2 \cdot 10^6$ for one level or $2^{42} \approx 2 \cdot 10^{12}$ considering the combined top and bottom level resolution. The AABBs are implicitly stored inside the transformations and can be obtained with $\mathrm{decompose}_{\mathrm{min,max}}$. While this is sufficient for static entities, we also must incorporate the motion of dynamic EDTs. To archive this, the AABBs must be extended to enclose the entire space the EDT will be covering within the upcoming time-step. The current implementation uses Forward Euler integration to predict the poses of the EDT in the consecutive frame as Equation 1

illustrates.

$$x_{\min} = \min(\text{decompose}_{\min}(\begin{bmatrix} \text{rot}({}^{\text{WS}}\underline{\theta}) \cdot \text{diag}(\underline{s}) & {}^{\text{WS}}\underline{x} \\ \underline{0} & 1 \end{bmatrix}), \text{decompose}_{\min}(\begin{bmatrix} \text{rot}({}^{\text{WS}}\underline{\theta}') \cdot \text{diag}(\underline{s}) & {}^{\text{WS}}\underline{x}' \\ \underline{0} & 1 \end{bmatrix})) \quad (1)$$

$$x_{\max} = \max(\text{decompose}_{\max}(\begin{bmatrix} \text{rot}({}^{\text{WS}}\underline{\theta}) \cdot \text{diag}(\underline{s}) & {}^{\text{WS}}\underline{x} \\ \underline{0} & 1 \end{bmatrix}), \text{decompose}_{\max}(\begin{bmatrix} \text{rot}({}^{\text{WS}}\underline{\theta}') \cdot \text{diag}(\underline{s}) & {}^{\text{WS}}\underline{x}' \\ \underline{0} & 1 \end{bmatrix}))$$

$$\text{with} \quad {}^{\text{WS}}\underline{\theta}' = \text{normalize}({}^{\text{WS}}\underline{\theta} + 0.5\Delta t \begin{bmatrix} {}^{\text{WS}}\dot{\underline{\theta}} & 0 \end{bmatrix} \cdot {}^{\text{WS}}\underline{\theta}), {}^{\text{WS}}\underline{x}' = {}^{\text{WS}}\underline{x} + \Delta t\, {}^{\text{WS}}\dot{\underline{x}}$$

By combining the AABBs of the current and the next frame, the resulting AABB ensures that all top-level queries gather the dynamic EDTs correctly. Figure 5 illustrates the procedure. For the BLAS we simply use the static AABB of the geometry. The algorithm further sorts the Morten Codes and organizes the nodes to an initial hierarchy as Karras (2012) describes. Our approach further adds a step before building the bottom-up AABB-hierarchy, which merges the BVH2 nodes together to BVH8 nodes in a bottom-up manner. Finally, the AABB-hierarchy is built as suggested.

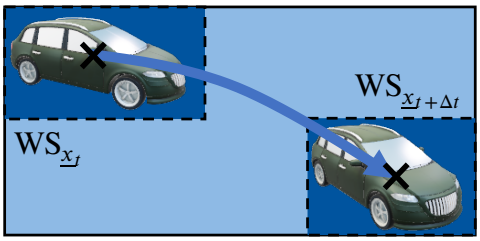

Figure 5: Extension of AABB related to movement of EDT.

The next stage is to develop the SDF for the geometries after establishing one major pillar of the ASs. While the build of zero-order SDFs is straightforward the generation of first-order SDFs requires a generalization of SDFs using Taylor Series as Bán and Valasek (2020) suggests. The approach can be summarized by building a local zero-order SDFs for a small area around each grid vertex and fit a plane to it by linear regression. To account for the distance to the vertex center, the sub-samples can further be weighted by their distance. The ensuing question of how to compute the process's zero-order SDF can be solved by using a Nearest Neighbor Query for distance and a Ray-Tracing Query for sign. To compute the sign, we use a heuristic proposed by Wright (2015), which cast rays evenly distributed over a sphere, centered at the grid vertex. We then compute the ratio α_{in} of triangles hit from behind, divided by the total number of rays traced. For $\alpha_{in} \geq \delta_{in}$ the location is considered inside. While the author suggested a $\delta_{in} = 0.5$, we found that a higher threshold of $\delta_{in} = 0.75$ improves the representation of concave corners.

Finally, the first-order SDFs can be used to update the local TLAS caches. We choose an SDF resolution of 128^3. For this purpose the GPUs rasterizer is leveraged to render the EDTs' first-order SDFs into local zero-order SDF. Since the rasterizer works on triangles, we use the triangulated AABBs from the EDTs. This gives us one invocation of the GPU program for each SDF vertex the AABB touches. We can further evaluate the first-order SDF at the vertex and write the value with an interlocked minimum operation into the zero-order SDF.

We can now proceed with the implementation of the queries after setting up the ASs.

5.1 Query Data

The input data must be set up before we can discuss the tracing procedure. Table 2 gives an overview of the data that the queries require as input and output. Because its input might be arbitrarily complicated and includes configuring a whole rendering pipeline, the Render Query is left out.

Table 2: Query input and output data. $\underline{x}$ are positions, $\underline{d}$ are directions, λ are ray direction factors e.i. $\underline{x}+\lambda\underline{d}$ is a point on the ray, $\underline{n}$ is a normalized vector, u,v are barycentric triangle coordinates, e and t are EDT id and triangle id respectively and N is the number of outputs, t_{start} is the start time.

Query	Input	Output
Ray-Tracing Query	$\underline{x},\underline{d},\lambda_{\min},\lambda_{\max},t_{\text{start}}$	λ,u,v,e,t
Cone-Tracing Query	$\underline{x},\underline{d},\alpha,\lambda_{\max},t_{\text{start}}$	λ,u,v,e,t
Nearest Neighbor Query	$\underline{x},r_{\max},t_{\text{start}}$	d,u,v,e,t
Region Query	$\underline{x}_{\min},\underline{x}_{\max},t_{\text{start}}$	$\{e_i,t_i,i\in[1\ N]\}$
Intersection Query	e	$\{e_i,\underline{x}_i,\underline{n}_i,\lambda_i,i\in[1\ N]\}$

5.2 Tracing and Nearest Neighbor Queries

Because the algorithmic structures of tracing and closest neighbor searches are similar, they will be discussed together. These queries unite a stack used to postpone the processing of child nodes, while processing the hierarchy in deep-first order. Here comes the compression of the BVH2 to an BVH8 into play, which allows to store up to 8, to be postponed child nodes, in only two 32 bit integer variables. The first simply holds the parent node id and the second holds on 3bit index and a 1bit active flag or each of the 8 children adding up to 32bit. Having the stack set up the rays, cones and positions can be tested against the eight children utilizing the GPU's vector register. For the rays, the intersection distances λ_i are computed for all the child boxes. For the cones, the child boxes are first enlarged with respect to the cone opening angle α and then processed equally to the rays. To find the nearest point the closest distances λ_i to the child nodes are computed. In the next step a sorting network is used to sort the candidates from the nearest to the furthest. Finally, the sorted child nodes are then pushed to the stack. In Case a leaf node is encountered one of three things must be done depending on the query type and top or bottom level. Leaves encountered while ray-tracing the TLAS are handled by transforming the ray into the child BLAS coordinate system. It is worth noting at this point that the transformation incorporates the velocity of the corresponding EDT, which the leaf node represents, utilizing the ray' start time. In the BLAS the leaves contain the triangles, which are then tested for intersection with the ray using Möller and Trumbore (1997) test. For cones a TLAS results similar to rays to a transformation into the child BLAS coordinate system, also aware of the motion. But from this point the tracing algorithm is exchanged to a SDF sphere tracing algorithm, leveraging the first-order SDFs. The sphere tracing algorithm steps through the SDF by sampling the nearest distance at a given point. Since this distance indicates the nearest surface, the cone can proceed inside the sphere, the distance spans, without intersecting anything. Figure 6 visualizes the algorithm. The Algorithm terminates when the spatial process the cone falls below a certain threshold or leaves the SDF. The last case is encountering leaf nodes while searching the nearest neighbor. In case of the TLAS the query point is also transformed to the child BLAS coordinate system, again incorporating the motion. In case of a BLAS leaf the nearest point on the respective triangle is computed. The candidates found while ray- or cone-tracing and doing nearest neighbor search are compared against the best one found so far, replacing it in case the new one found is closer.

6 APPLICATIONS

In the application section we are going to show how the RTSTDB can be used to simulate Digital Twins. The first application showcases the difference between temporal aware and unaware sensor simulation. The test scene consists of a laser scanner mounted to a car shown in Figure 7. The car is waiting at a crossing and perceives meanwhile the cars passing the crossing from the right. The laser scanner has a scanning frequency of 1Hz with 140° horizontal and 40° vertical field of view with a scanning resolution of 1920x1080.

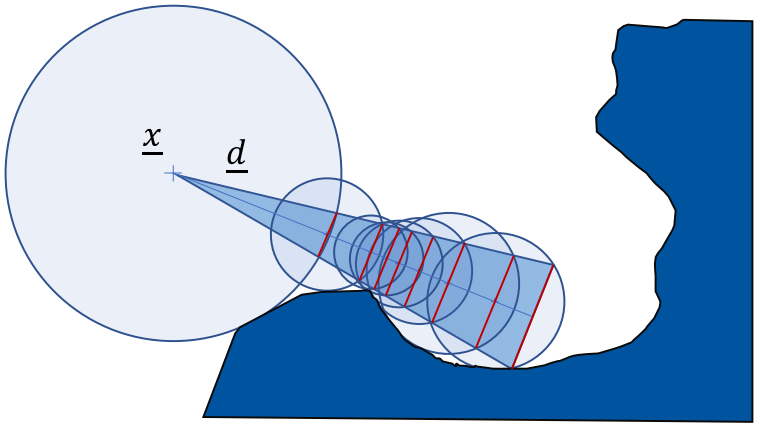

Figure 6: Cone tracing.

Figure 7: Overview of the laser scanner scene. Image generated using the path tracer provided by the RTSTDB.

Since the scanner requires 1s to scan from the left to right, the car it perceives will be at different locations depending on the time the individual scans are done. Figure 8 illustrates the effect of the car's movement by comparing the temporal aware with the temporal unaware ray-tracing. While in the top right image the temporal dependency is neglected and the car' perceived dimensions do not change, in the middle and bottom right images the real-time is done temporally aware resulting in the car being slightly elongated or compressed. Even though the effect is slightly exaggerated by using a scan rate of only 1Hz, the effect cannot be neglected to correctly simulate dynamic EDTs, which underlines the requirement for RTSTDB.

Another challenging task is the simulation of the fanning-out property of real laser-beams caused by physical limitation of real laser emitter. While this can be approximated via ray-tracing using multiple rays, the approach suffers from two shortcomings. The first is that tiny objects can be missed over large distances and the second problem is low efficiency due to the high number of rays. This can be overcome using cone-tracing, which correctly represents the fanning-out of the laser beam. One example task could be a docking maneuver in space, where a SpaceX Dragon capsule tries to dock the International Space Station (ISS) as shown in Figure 9.

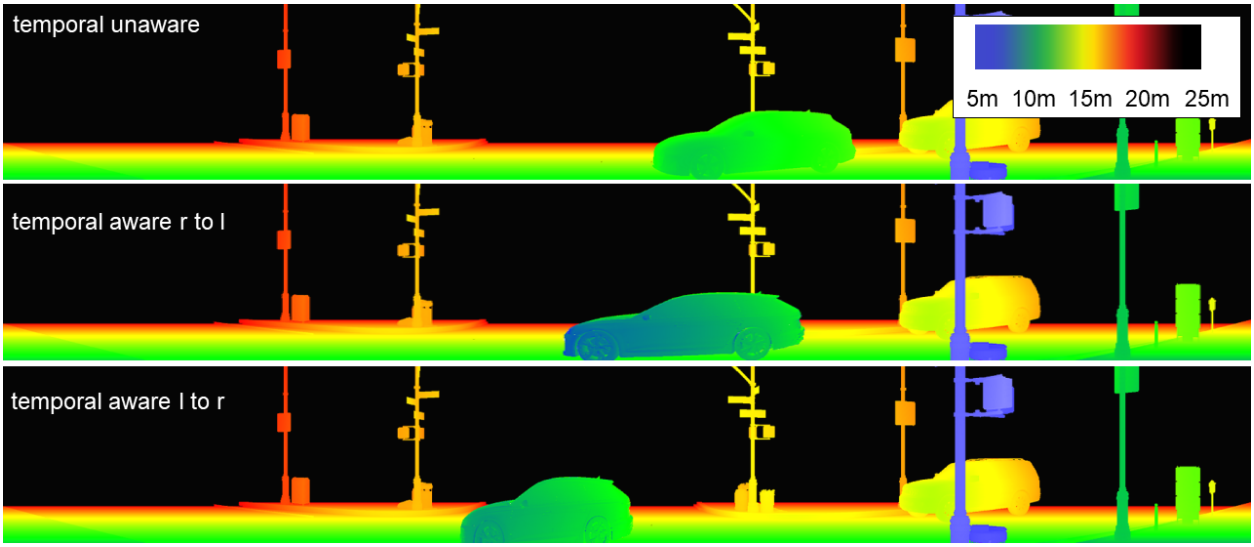

Figure 8: Comparison of temporal aware vs. unaware ray-tracing using a simulated laser scanner.

To determine the location of the docking port, the shuttle uses a laser scanner to detect retro reflectors around the port. Figure 10 compares the simulation results of the Ray-Tracing and Cone-Tracing approaches. The virtual sensor has a field of view of 40°x40° with a resolution of 1080x1080 pixels and a cone opening of 0.02°. The graphics further proves that the ray-tracing approach cannot reliably detect the docking port as the distance exceeds 200m even though using 4096 samples per pixel. The Cone-Tracer on the other side detects the docking port at every distance using only one sample per pixel. This was to be expected considering the Nyquist criterion, which requires the sampling frequency to be twice the signal frequency for correct reconstruction. This translates to Equation 2 with fov_{max} the maximum field of view, S_{min} the minimum object size, d_{max} the maximum distance the object should be detected at, N_{sample},N_{sub} are sensor' pixel count and sub sample count respectively.

$$\frac{\text{fov}_{\text{max}}}{\text{atan}(\frac{S_{\text{min}}}{2 \cdot d_{\text{max}}})} \leq N_{\text{sample}} \cdot N_{\text{sub}} \tag{2}$$

Another consideration is the duration it takes to ray-trace such an amount of samples. While the cone tracing requires a constant time of 5ms to trace one sample, the ray-tracing exceeds this time taking more than 2 samples. The approach using 4096 samples per pixel requires even more than 10s, which does not fit into real-time application.

7 SUMMARY

In this work we identified the requirements EDTs pose on the simulation' database to be real-time capabilities, spatial organizability of data and temporal awareness in data representation. These requirements were aligned with state of art, which does not provide a holistic solution. By introducing the RTSTDB architecture we could provide a viable solution combining multiple approaches from the state of art into one design. The definition of the interface was discussed and the key implementation details were shown. Finally we showed a working application of our approach by providing two use cases that exhibit the identified requirements. The first example successfully illustrates the incorporation of spatio-temporal properties of EDTs in the scenario. The second use-case highlights the flexibility and real-time abilities of our approach. Further research effort could target different geometry representations for cone-tracing or collision-detection. Another direction could aim on distributing the database over a cluster or network of servers, splitting-up the workload.

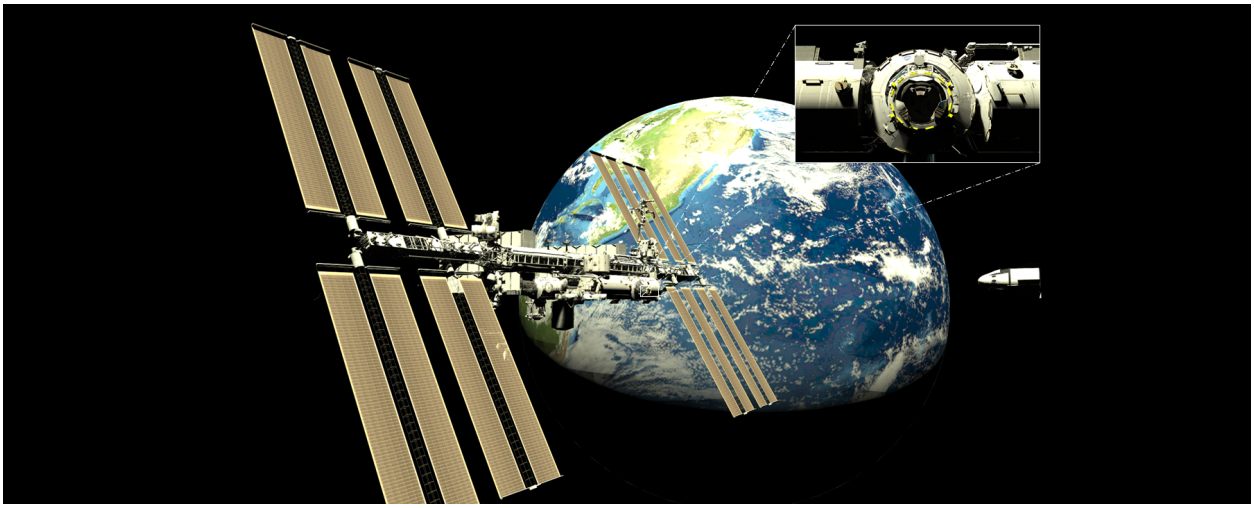

Figure 9: Docking maneuver of a SpaceX Dragon capsule using laser scanners to detect retro reflector at the docking port. Image generated using the path tracer provided by the RTSTDB.

ACKNOWLEDGMENTS

This work shows results of the projects “ViTOS-II” and “ViTOS-3”, supported by the German Aerospace Center (DLR) with funds of the German Federal Ministry of Economics and Technology (BMWi), support code 50 RA 1810 and 50 RA 2120.

REFERENCES

Balusamy, B., N. A. R, S. Kadry, and A. H. Gandomi. 2021. *NoSQL Database*.

Benoit, T., S. Moten, and Y. Lemmens. 2017. “Real-time Physics-based Simulation of Mechanisms and Systems”. 1–2.

Benthin, C., I. Wald, S. Woop, and A. T. Áfra. 2018. “Compressed-leaf Bounding Volume Hierarchies”. Association for Computing Machinery.

Bán, R., and G. Valasek. 2020, 3. “First Order Signed Distance Fields”.

Emde, M., and J. Rossmann. 2013. “Validating a Simulation of a Single Ray Based Laser Scanner Used in Mobile Robot Applications”. 55–60.

Ester, M., M. Ester, H.-P. Kriegel, and J. Sander. 1999. “Knowledge Discovery in Spatial Databases”. In *Advances In Artificial Intelligence, 23rd Annual German Conference On Artificial Intelligence* 61:61–74.

Garg, P., V. Saxena, and S. K. Singh. 2016. “Optimizing Spatial Database Performance for Handling Moving Objects”. 1–3.

Gilbert, E. G., D. W. Johnson, and S. S. Keerthi. 1987. “Fast Procedure for Computing the Distance between Complex Objects in Three Space”. 1883–1880: IEEE.

Grieves, M. 2002, 3. “Completing the Cycle: Using PLM Information in the Sales and Service Functions”. This was the first public presentation of the Digital Twin Model. It didn’t have a name. It was simply ”Conceptual Ideal for PLM.” It was later presented at the University of Michigan Lurie Engineering Center at an organizing meeting for the PLM Consortium.

Grieves, M. 2011, 3. *Virtually Perfect: Driving Innovative and Lean Products through Product Lifecycle Management*. First Mentioning of Digital Twin.

Grieves, M. 2016, 3. “Origins of the Digital Twin Concept”. *Florida Institute of Technology* 8.

Jürgen, M. R., and Schluse. 2020. *Experimentierbare Digitale Zwillinge im Lebenszyklus Technischer Systeme*. Springer Berlin Heidelberg.

Karras, T. 2012. “Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees”. 33–37: Eurographics Association.

Karras, T., and T. Aila. 2013. “Fast Parallel Construction of High-Quality Bounding Volume Hierarchies”. 89–99: Association for Computing Machinery.

Koschier, D., C. Deul, and J. Bender. 2016. “Hierarchical Hp-Adaptive Signed Distance Fields”. 189–198: Eurographics Association.

Kuroki, S., A. Makinouchi, and K. Ishizuka. 1997. “Towards a Spatio-temporal OQL for the Four Dimensional Spatial Database System Hawks”. 142–147.

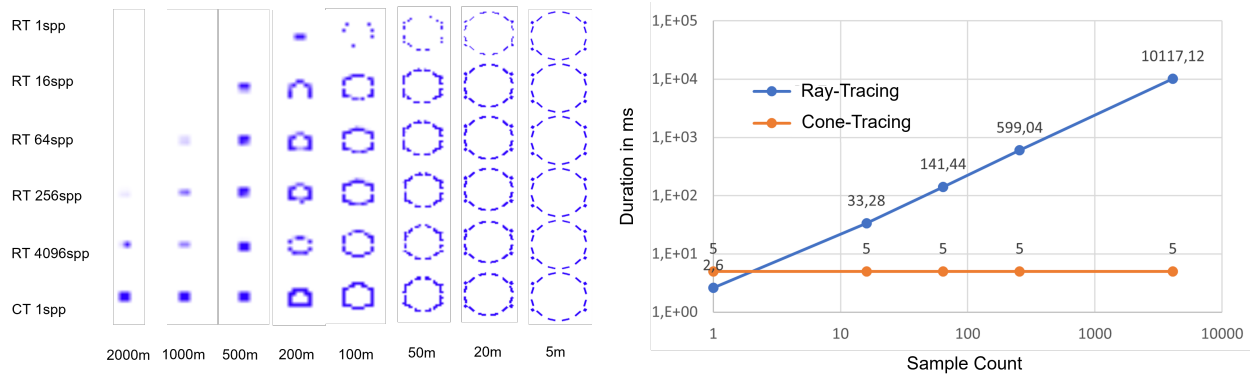

Figure 10: Laser scanner results of the docking maneuver. On the left, the first four rows show the Ray-Tracing results using 1, 16, 64, 256, 4096 samples per pixel. The last row shows the results of the Cone-Tracing using only one sample per pixel. On the right the frame duration is plotted over the sample count. The ISS model has a triangle count of 250K.

Lee, B.-R. 2021. "IEEE Standard for Head-mounted Display (HMD)-based Virtual Reality(VR) Sickness Reduction Technology". *IEEE Std 3079-2020*:1–74.

Lin, K.-J., and S. H. Son. 1994. "Real-time Databases: Characteristics and Issues". 113–116.

Möller, T., and B. Trumbore. 1997, 1. "Fast, Minimum Storage Ray-triangle Intersection". *Journal of Graphics Tools* 2:21–28.

Rahnamai, K., and J. Rowlands. 2017. "State Space Design Cycle and Hardware in the Loop Testing and Verification". 603–606.

Schluse, M., M. Priggemeyer, L. Atorf, and J. Rossmann. 2018. "Experimentable Digital Twins—streamlining Simulation-based Systems Engineering for Industry 4.0". *IEEE Transactions on Industrial Informatics* 14:1722–1731.

Shekhar, S., S. Chawla, S. Ravada, A. Fetterer, X. Liu, and C.-T. Lu. 1999. "Spatial Databases-accomplishments and Research Needs". *IEEE Transactions on Knowledge and Data Engineering* 11:45–55.

Thieling, J., S. Frese, and J. Roßmann. 2021. "Scalable and Physical Radar Sensor Simulation for Interacting Digital Twins". *IEEE Sensors Journal* 21:3184–3192.

Tian, Y., Y. Ji, and J. Scholer. 2015. "A Prototype Spatio-temporal Database Built on Top of Relational Database". 14–19.

van den Bergen, G. 2003, 10. *Collision Detection in Interactive 3D Environments*. CRC Press.

Wright, D. 2015. "Dynamic Occlusion with Signed Distance Fields".

Yu, H., X. Li, L. Yuan, and X. Qin. 2021. "Efficient Spatio-temporal-data-oriented Range Query Processing for Air Traffic Flow Statistics". 1303–1310.

Zheng, Y. C., Y. Deng, and Q. M. Zhu. 2014. "Ship Damage Control as a Service Based on Spatio-temporal Database". 500–504.

AUTHOR BIOGRAPHIES

MORITZ ALFRINK received his double master degree in electrical engineering from the RWTH Aachen, Germany and the KTH Royal Institute of Technology, Sweden, in 2018. His current field of research at the Institute of Man-Machine-Interaction in Aachen, Germany is simulation technology and computer graphics. Besides his research he develops a computer graphics engine in C++ and creates robots. His e-mail address is alfrink@mmi.rwth-aachen.de.

JUERGEN ROßMANN is a professor with the faculty of electrical engineering at RWTH Aachen University and head of the Institute for Man-Machine Interaction. After graduating from the Universities of Dortmund and Bochum, he worked as a research assistant, then as a group leader and finally as a department head at the Institute of Robotics Research at the University of Dortmund. His work in the fields of space and industrial robotics, automation technology and virtual reality has been awarded more than 15 different prizes. He is a visiting professor for robotics and computer graphics at the University of Southern California since 1998. Furthermore, he is a member of the board of Directors of the Dortmund Institute for Research and Transfer (RIF) as well as managing director of VEROSIM GmbH. His email address is rossmann@mmi.rwth-aachen.de.